A Local Search Modeling for Constrained Optimum Paths Problems (Extended Abstract)

Quang Dung Pham

Université catholique de Louvain
B-1348 Louvain-la-Neuve, Belgium

quang.pham@uclouvain.be

Yves Deville

Université catholique de Louvain
B-1348 Louvain-la-Neuve, Belgium

yves.deville@uclouvain.be

Pascal van Hentenryck

Brown University, Box 1910
Providence, RI 02912, USA

pvh@cs.brown.edu

Constrained Optimum Path (COP) problems appear in many real-life applications, especially on communication networks. Some of these problems have been considered and solved by specific techniques which are usually difficult to extend. In this paper, we introduce a novel local search modeling for solving some COPs by local search. The modeling features the compositionality, modularity, reuse and strengthens the benefits of Constrained-Based Local Search [1]. We also apply the modeling to the edge-disjoint paths problem (EDP). Computational results show the significance of the approach.

1 Introduction

Constrained Optimum Path (COP) problems, where optimum path from origin to destination satisfying additional constraints must be found, appear in many real-life applications, especially on communication and transportation networks. These problems have attracted considerable attention from different research communities: operations research, telecommunications because of its various applications (see [2] and the references therein).

Most of COP problems are NP-hard. Some instances can be solved efficiently by specific techniques, for instance, branch and bound using a Lagrangian-based bound [3], vertex-labeling [4], etc. These techniques seem to be sophisticated and depend on particular constraints and objective functions to be optimized. Moreover, they are difficult to extend, for example, when we face with generalized problems where more constraints are required to be satisfied. We propose in this paper a novel approach for modeling and solving some COP problems by local search where the desired paths are elementary (i.e. no repeated nodes). The objective of this work is to extend the LS(Graph & Tree) framework [5] by the design and the implementation of abstractions allowing to easily model and solve some COP problems using this approach. The computational model features compositionality, modularity, reuse, and strengthens the local search modeling benefits of Constraint-Based Local Search [1] which provides the separation of concerns.

The proposed computational model has been applied to the EDP problem. Computational results show the significance of the approach.

2 Modeling paths with spanning trees

We introduce in this section the proposed approach for modeling COP problems with spanning trees. We first give somes definitions and notations over graphs. Our framework considers both directed and undirected graphs but for simplicity, we consider in this presentation only undirected graphs.

Given an undirected graph g, we denote $V(g)$, $E(g)$ respectively the set of nodes and the set of edges of g.

In this paper, we only consider elementary paths, henceforth we use the word "path" instead of "elementary path" if there is no ambiguity. A graph is connected if and only if there exists a path from u to $v, \forall u, v \in V(g)$. A tree is an undirected connected graph containing no cycles. A spanning tree tr of an undirected connected graph g is a tree spanning all the nodes of g: $V(tr) = V(g)$ and $E(tr) \subseteq E(g)$. A tree tr is called a rooted tree at r if the node r has been designated the root. Each edge of tr is implicitly oriented towards the root. If the edge (u, v) is oriented from u to v, we call v the father of u on tr.

The key decision design is inspired from the following observation: Given a rooted tree tr whose root is t, the path from a given node s to t on tr is unique. An update of tr will generate a new rooted tree which may induce a new path from s to t on this tree.

Given an undirected graph g and a node $r \in V(g)$, *VarRootedSpanningTree*(g, r) (also called rooted spanning tree variable) is a concept representing a dynamic spanning tree of g. The spanning tree is rooted at r.

In order to model a COP problem in an undirected graph g in which the source and the target of the desired path are respectively $s, t \in V(g)$, we use *VarRootedSpanningTree*(g, s, t) which is *VarRootedSpanningTree*(g, t) with a node s designated as the source node. Each instance tr of *VarRootedSpanningTree*(g, s, t) specifies a unique path from s to t on g. Henceforth we use s to denote the source node of any rooted spanning tree of the given graph and the path from s to the root of a rooted spanning tree tr on tr is called the path induced by tr if there is no ambiguity.

The main avantage of using rooted spanning tree for modeling paths instead of using explicit paths representation (i.e. a sequence of nodes) is the simplification of neighborhood computation. The tree structure constains rich information that induces directly path structure from a node s to the root. A simple update over that tree (i.e. an edge replacement which is detailed in Section 3) will induce a new path from s to the root.

3   Neighborhood

Given an instance tr of *VarRootedSpanningTree*(g, s, t), we show how to change tr in order to generate a new rooted spanning tree tr' of g which induces a new path from s to t on g.

Given an undirected graph g, an instance tr of *VarRootedSpanningTree*(g, s, t), an edge $e = (u, v)$ such that $e \in E(g) \setminus E(tr)$ is called *replacing* edge of tr. We denote $rpl(tr)$ the set of *replacing* edges of tr. Given $e \in rpl(tr)$, an edge e' that belongs to the path between two endpoints of e on tr is called *replacable* edge of e. We denote $rpl(tr, e)$ the set of *replacable* edges of e. Intuitively, a *replacing* edge e is an edge that is not in the tree tr but that can be added to tr (this edge insertion creates a cycle C when we ignore orientations of edges of tr), and all edges of this cycle except e are *replacable* edges of e.

Given an undirected graph g, an instance tr of *VarRootedSpanningTree*(g, s, t), e and e' are respectively *replacing* edge of tr and *replacable* edge of e, we define the following edge replacement action:

1. Insert the edge $e = (u, v)$ to tr. This creates an undirected graph g with a cycle C containing the edge e'.

2. Remove e' from g.

After taking above edge replacement action, we obtain a new rooted spaning tree tr' of g. We denote $tr' = rep(tr, e', e)$. The neighborhood of tr is

$$N(tr) = \{tr' = rep(tr, e', e) \mid e \in rpl(tr), e' \in rpl(tr, e)\}$$

It is easy to observe that two different spanning trees tr_1 and tr_2 rooted at t of an undirected graph g may induce the same path from s to t on g $(s, t \in V(g))$. The neighborhood $N(tr)$ must then be reduced
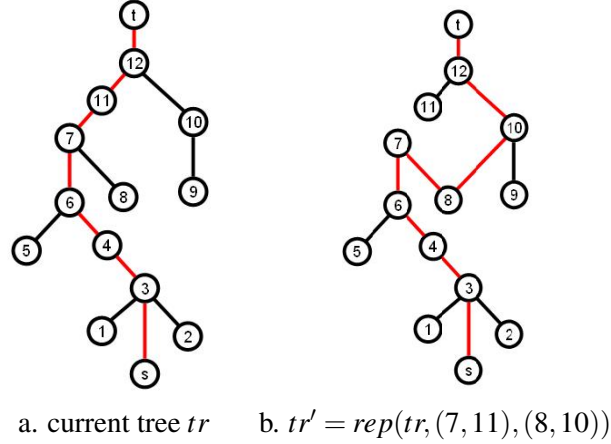
a. current tree tr b. $tr' = rep(tr, (7,11), (8,10))$

Figure 1: Example of basic move

a. current tree tr b. $tr' = rep(tr, (7,11), (8,10), (3,4), (1,5))$
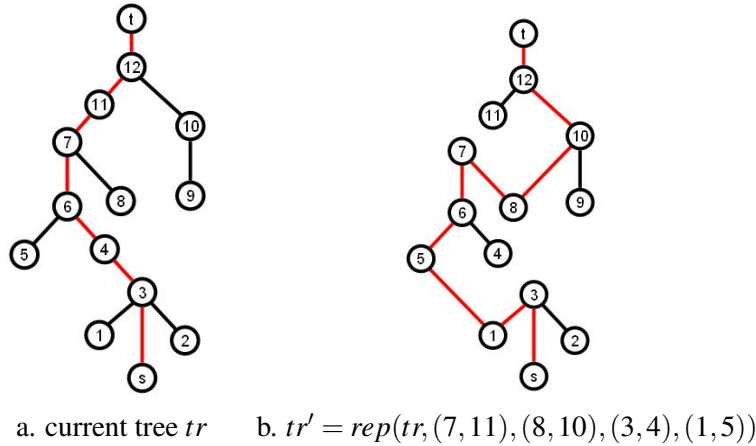
Figure 2: Example of complex move

such that the new tree induces a new path from *s* to *t*. This reduction is described in [6]. The action $rep(tr, eo, ei)$ is called a basic move. Figure 1 gives an example of basic move.

It is possible to consider more complex moves by applying a set of independent basic moves. Two basic moves are independent if the execution of the first one does not affect the second one and vice versa. The execution order of these basic moves does not affect the final result. Figure 2 gives an example of complex move.

4 COMET **implementation**

We extend the LS(Graph & Tree) framework by implementing some *GraphInvariants*, *GraphConstraints* and *GraphObjectives* (see [5] for more detail) for modeling and solving some COP problems. *GraphInvariant* is a concept representing objects which maintain some properties of a dynamic graph[1]

---

[1]dynamic graph is a graph that can be changed e.g., by the removal or the insertion of vertices, edges.

```
1.    LSGraphSolver ls();
2.    VarRootedSpanningTree tr(ls,g,s,t);
3.    PreferredReplacingEdges prefReplacing(tr);
4.    PreferredReplacableEdges prefReplacable(tr);
5.    forall(ei in prefReplacing.getSet())
6.        forall(eo in prefReplacable.getSet(ei))
7.            d = C.getReplaceEdgeDelta(tr,eo,ei);
```

Figure 3: Exploring the basic neighborhood

(for instance, the sum of weights of all the edges of a graph, the diameter of a tree, etc.). *GraphConstraint* and *GraphObjective* are concepts describing differentiable objects which maintain some properties (for instance, the number of violations of a constraint or the value of an objective function) of a dynamic graph. The main feature of *GraphConstraint* and *GraphObjective* is the interface, allowing to query the impact of local moves (modification of the dynamic graph) on these properties. Some *GraphConstraint*s and *GraphObjective*s have been designed and implemented over *VarRootedSpanningTree*. For instance, PathCostOnEdges(tr,k) [2] is an abstraction representing the total weight accumulated along the path induced by tr, MinEdgeCost(tr,k), MaxEdgeCost(tr,k) are abstractions representing the minimal and maximal weight of edges on the path induced by tr. NodesVisited(tr,S) is an abstraction representing the number of nodes of S visited by the path induced by tr. These abstractions are example of *GraphObjective*s which are fundamental when modeling Constrained Optimum Path problems. For example, in QoS, we consider shortest path from an origin to a destination with constraints over bandwidth which is defined to be the minimum weight of edges on the specified path. The framework provides flexibility for modeling various Constrained Optimum Path problems. We can easily combine (with +,-,* operator) and state basic constraints (for instance, $<=, >=, ==$) over these abstractions. PathEdgeDisjoint(tr) is another *GraphConstraint* which is defined over an array of paths (the ith path is induced by tr[i]) and specifies that these paths are mutually edge-disjoint.

   In order to illustrate the modeling, we give a snippet (see Figure 3) which explores the basic neighborhood. Line 1 initializes a LSGraphSolver object ls which manages all the *VarGraph*, *VarTree*, *GraphInviant*s, *GraphConstraint*s and *GraphObjective*s and relations (dependency graph) between these objects. Line 2 declares and initializes randomly a VarRootedSpanningTree tr rooted at t of the input graph g which represents the path from the source node s to t. prefReplacing and prefReplacable are *GraphInvariant*s which maintain the set of *preferred replacing* edges[3] and *preferred replacable* edges (lines 3-4). Lines 5-7 explore the basic neighborhood and evaluate the quality of moves with respect to a *GraphConstraint* C. The getReplaceEdgeDelta (line 7) method returns the variation of the number of violations of C when the *preferred replacable* edge eo is replaced by the *preferred replacing* edge ei on tr.

5   Application: The EDP problem

Given an undirected graph $G = (V, E)$, and a set $T = \{< s_i, t_i > | \ s_i \neq t_i \in V\}$ representing a list of commodities ($\sharp T = k$). EDP consists of finding a maximal cardinality set of mutually edge-disjoint paths

---

[2]tr is a *VarRootedSpanningTree*, k is the index of the considered weight on edges.

[3]edges used for the edge replacements which return new trees inducing new paths from *s* to *t*.

```
void stateModel{
1.    LSGraphSolver ls();
2.    VarRootedSpanningTree tr[i in 1..k](ls,g,s[i],t[i]);
3.    PathEdgeDisjoint ed(tr);
4.    ls.close();
5.}
```

Figure 4: The Modeling for the EDP problem

from s_i to t_i on G ($< s_i, t_i > \in T$). In [7], a Multi-start Simple Greedy algorithm is presented as well as an ACO algorithm. The ACO is known to be state-of-the-art for this problem. We propose a local search algorithm using the modeling approach. The model is given in Figure 4 where line 2 initializes an array of k `VarRootedSpanningTrees` representing k paths between commodities. The edge-disjoint constraint ed is which is defined over paths from `s[i]` to `t[i]` on `tr[i]` (i = 1, 2, ..., k) stated in line 3. In [7], the following criterion is introduced which quantifies the degree of non-disjointness of a solution $S = \{P_1, P_2, ...P_k\}$ (P_j is a path from s_j to t_j):

$$C(S) = \sum_{e \in E}(max\{0, \sum_{P_j \in S} \rho^j(S,e) - 1\})$$

where $\rho^j(S,e) = 1$, if $e \in P_j \in S$ and $\rho^j(S,e) = 0$, otherwise.

The number of violations of the *PathEdgeDisjoint*$(P_1, P_2, ..., P_k)$ constraint in the framework is defined to be $C(\{P_1, P_2, ..., P_k\})$ and the proposed local search algorithm tries to minimize this criterion. From a solution which is normally a set of k non-disjoint, a feasible solution to the EDP problem can be extracted by iteratively removing the path which has most edges in common with other paths until all remaining paths are mutually edge-disjoint as suggested in [7]. In our local search model, we extend that idea by taking a simple greedy algorithm over the remaining paths after that extraction procedure in hope of improving the number of edge-disjoint paths.

The main idea for the search is to try different moves to get a first improvement: 1-move or 2-move over one `VarRootedSpanningTree`, two 1-moves at hand over two `VarRootedSpanningTrees`.

For the experimentation, we re-implemented in `COMET` the Multi-start Greedy Algorithm (MSGA) and the ACO (the extended version) algorithm which are described in [7], and compare them with our local search model. The instances experimented (graphs including commodities) in the original paper [7] is not available at the moment (except some graphs). We base on the instances generation description in the paper [7] and generate new instances as follows. We take 4 graphs from [7]. For each graph, we generate randomly different sets of commodities with different sizes depending on the size of the graph: for each graph of size *n*, we generate randomly 20 instances with 0.10*n, 0.25*n and 0.40*n commodities. In total, we have 240 problems instances. Due to the huge complexity of the problem, we execute each problem instance once with a time limit of 30 minutes for each execution. Experimental results are shown in Table 1. The time window for the MSGA and the ACO algorithms are also 30 minutes. The Table reports the average values of the objective function of the best solutions found and the average of times for obtaining these best solutions of 20 instances (a graph $G = (V, E)$ and a set of $r*|V|$ commodities, $r = 0.10, 0.25, 0.40$). The Table shows that our local search model gives competitive results in comparison with the MSGA and the ACO algorithms. In comparison with the MSGA, our local search model find better solutions in 217/240 instances while the MSGA find better solutions in 4/240

instance	com.	MSGA		ACO		Local search	
		$\bar{q}$	$\bar{t}$	$\bar{q}$	$\bar{t}$	$\bar{q}$	$\bar{t}$
mesh25x25.bb	62	36.95	546.854	31.1	880.551	**38.85**	1165.47
	156	44.65	863.007	47.5	965.921	**55.5**	1082.78
	250	50.5	672.962	60.5	972.396	**67.95**	967.087
mesh15x15.bb	22	20.55	517.601	18.6	500.812	**21**	384.828
	56	27.15	651.27	28.35	988.782	**30.3**	485.693
	90	31	797.534	34.55	746.96	**36.05**	435.308
bl-wr2-wht2.10-50.rand.bb	50	18.7	688.651	19.6	201.235	**20.05**	228.382
	125	27.2	643.51	31.15	338.446	**31.2**	241.047
	200	36.6	625.138	41.55	164.783	**41.7**	202.186
bl-wr2-wht2.10-50.sdeg.bb	50	18.65	470.26	19.75	223.396	**20.1**	311.887
	125	28.1	662.916	31.55	163.151	**31.85**	357.25
	200	33.3	487.999	38.05	217.362	**38.25**	178.417

Table 1: Experimental results of EDP problem

instances. On the other hand, in comparison with the ACO model, our local search model find better solutions in 144/240 instances while the ACO model find better solutions in 11/240 instances.

6 Conclusion and future work

We introduce in this paper a novel local search modeling for Constrained Optimum Path problems on graphs. The objective here is to give a high-level modeling framework for implementing some COP problems which strengthens the benefits modeling of CBLS and features compositionality, modularity and reuse. The modeling provides a clean seperation of concerns: The modeling component and the search component are independent. On one hand, it is easy to add new constraints and to modify or remove existing ones, without having to worry about the global effect of these changes. On the other hand, programmers can experiment with different heuristic and metaheuristics without affecting the problem modeling. The modeling is based on *VarRootedSpanningTree*(g, s, t) concept inspiring the observation that each tree induces a unique path between two specified nodes and a update over this tree satisfying additional contraints generates a new tree which induces a new path between these nodes. Various neighborhoods for have been defined. The abstraction is implemented by extending the LS(Graph & Tree) in COMET. The modeling has been experimented on the resources constrained shortest path problem and the edge-disjoint paths problem which show the significance of the framework.

7 Bibliography

References

[1] Pascal Van Hentenrych and Laurent Michel (2005): *Constraint-based local search. The MIT Press.*

[2] W. Matthew Carlyle and R. Kevin Wood (2003): *Lagrangian Relaxation and Enumeration for Solving Constrained Shortest-Path Problems*. Proceedings of the 38th Annual ORSNZ Conference.

[3] J. E. Beasley and N. Christofides (1989): *An Algorithm for the Resource Constrained Shortest Path Problem. Network, vol. 19*, pp. 379–394.

[4] I. Dumitrescu and N. Boland (2003): *Improved Preprocessing, Labeling and Scaling Algorithms for the Weight-Constrained Shortest Path Problem. Networks, 42*, pp. 135–153.

[5] Quang Dung Pham and Yves Deville and Pascal Van Hentenryck (2009): *LS(Graph & Tree): A Local Search Framework for Constraint Optimization on Graphs and Trees*. Proceedings of the 24th Annual ACM Symposium on Applied Computing (SAC'09).

[6] Quang Dung Pham and Yves Deville and Pascal Van Hentenryck: *A Local Search Modeling for Constrained Optimum Paths Problems*. INGI Technical Report, UCLouvain, Belgique, August 2009.

[7] M. Blesa and C. Blum (2007): *Finding edge-disjoint paths in networks: An ant colony optimization algorithm. Journal of Mathematical Modelling and Algorithms, 6(3)*, pp. 361–391.